A hybrid approach for biobjective optimization

Stidsen, Thomas Jacob Riis; Andersen, Kim Allan

Published in:
Discrete Optimization

Link to article, DOI:
10.1016/j.disopt.2018.02.001

Publication date:
2018

Document Version
Peer reviewed version

A hybrid approach for biobjective optimization

Thomas Stidsen[a,*], Kim Allan Andersen[b]

[a]*DTU-Management, Technical University of Denmark*
[b]*Department of Economics and Business Economics, Aarhus University*

Abstract

A large number of the real world planning problems which are today solved using Operations Research methods are actually multiobjective planning problems, but most of them are solved using singleobjective methods. The reason for converting, i.e. simplifying, multiobjective problems to singleobjective problems is that no standard multiobjective solvers exist and specialized algorithms need to be programmed from scratch.

In this article we will present a hybrid approach, which operates both in decision space and in objective space. The approach enables massive efficient parallelisation and can be used to a wide variety of biobjective Mixed Integer Programming models. We test the approach on the biobjective extension of the classic traveling salesman problem, on the standard datasets, and determine the full set of nondominated points. This has only been done once before [1], and in our approach we do it in a fraction of the time.

Keywords: biobjective optimization; mixed integer programming; traveling salesman problem; branch-and-cut algorithm

1. Introduction

Most real-world problems which are optimized using Operations Research (OR) methods are actually multiobjective. Only very seldom do the OR practitioners consider their optimization problem as multiobjective optimization problems, i.e. in most cases the objectives are simply summed, possibly with weights thus emphasizing the most important objectives. While this approach may work in practice, it is problematic since it ignores a multitude of other (Pareto) optimal solutions. While there has been research going on in multiobjective optimization for decades, an increased interest in exact solution of multiobjective optimization problems has flourished in the last decade (i.e. since 2006): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. This research has however not yet lead to general practical solvers of multiobjective Mixed Integer Programming (MOMIP) models.

Generally speaking there have been two different types of approaches: Criteria Space Search (CSS) algorithms and Decision Space Search (DSS) algorithms. We will briefly describe the most important CSS algorithms and DSS algorithms below.

*I am corresponding author
Email addresses: thst@dtu.dk (Thomas Stidsen), kia@econ.au.dk (Kim Allan Andersen)

1.1. Criteria space search algorithms

CSS algorithms are iterative algorithms which utilize a series of singleobjective optimizations, where extra constraints are added to the MOMIP models criteria space. The two "classic" CSS algorithms are the ϵ-constraint method [15] and the two-phase method [16]. The ϵ-constraint method starts by finding one of the two lexicographic points. This point is then ruled out by a constraint on one of the criteria space variables and another (lexicographic) optimization leads to the next point of the Pareto front. The two-phase method starts with the two lexicographic points. In the first phase, the nondominated extreme (and possibly some non-extreme) points of the Pareto front are found by solving a series of singleobjective MIP's. In the second phase, a series of "triangles" in criteria space are iteratively searched for solutions, again using singleobjective MIP's or problem-dependent enumeration methods.

While the CSS algorithms have been known for decades modern versions have been constructed which seems very promising, e.g. [3, 6].

1.2. Decision space search algorithms

DSS algorithms are (basically) branch-and-bound algorithms, tweaked to work with more than one objective. This requires a re-definition of the fathoming rules, new types of branching and new versions of pre-processing, of probing etc. The cutting plane generation is however **not** changed! Building an efficient branch-and-cut algorithm from scratch is a monumental task, which only very few researchers could possible do and the developed algorithm would most likely still be seriously inferior to standard solvers like CPLEX or Gurobi, if applied to singleobjective MIP models. This approach has been attempted a few times in [17, 18]. Recently, there has however been approaches which utilize the so-called callback procedures in modern solvers such as CPLEX or Gurobi [10, 13]. These approaches overcomes **some** of the issues.

1.3. Which approach is the best? CSS or DSS?

The question of which approach is the best, i.e. can solve the largest MOMIP models fastest is in our opinion open. More research is required and both approaches have their advantages and disadvantages: The CSS algorithms can utilize the fast development of new solver versions, which promise continuous improvement. Their main disadvantage is that they solve a large set of almost alike MIP models, and they will most likely process the same intermediate solutions, i.e. nodes in their branch-and-bound tree, many times. The DSS algorithms on the other hand suffer from reduced strength in the fathoming process and the extra requirement of integer branching, see Section 3.2. On the other hand, they will ideally only see a node once.

In our opinion the question is not CSS or DSS, but **CSS and DSS!** We believe that the paradigms should be mixed to improve performance, thus creating hybrid methods. Several papers has already been proposed that combine both approaches, of which we just name a few. For instance, [19] embeds dynamic programming into a two phases method, using bound sets to discard triangles to be searched. The method has been applied to the biobjective knapsack problem with promising results. [20] proposes a biobjective local branching method

for mixed binary biobjective programming problems with promising results, although the CPU times are rather high, and [21] presents an algorithm that can determine all efficient points for the biobjective integer minimum cost flow problem, using a two-phases method combined with a ranking algorithm in the second phase. In our previous paper [10] we have also demonstrated the advantage of combining CSS and DSS, where a special branching method, called Pareto branching, is introduced. This allows branching in the criteria space, thus creating a hybrid approach. In our opinion this is a viable approach: To "enrich" DSS algorithms, with features from CSS algorithms, to speed up performance.

1.4. Parallel approaches

Parallelization of time-consuming algorithms is a classic way of achieving faster results. This approach has gained more importance in the last decade, where the clock-frequency has not increased significantly, but the number of cores on each chip has. Today, access to a large cluster of e.g. linux computers is relatively easy, cheap and common. This also means that solvers today like CPLEX or Gurobi focus a lot on parallel execution of especially branch-and-bound algorithms. Given that MOMIP models are significantly harder to solve than standard MIP models, parallelization of MOMIP solvers is a pretty obvious approach to improve the speed of MOMIP solvers. Parallelization of singleobjective branch-and-bound solvers, is well understood and implemented, but only for limited parallelization, e.g. less than 100 processors. For massive parallelization, communication between the different processors becomes problematic (they have to exchange new incumbent values).

1.5. Contributions

During the last two decades the interest in developing branch-and-bound methods for MOMIP problems has grown. In particular, a number of methods have been designed for multiobjective integer problems, where the integers are required to be binary, see for example [17, 22, 23, 24]. Recently, approaches has been suggested for biobjective mixed integer programs [14] where the integers are required to be binary, for general biobjective mixed integer programs [13], and for general biobjective integer programs [9]. In particular, the method presented in [13] makes use of advanced fathoming rules, and the method presented in [9] develop a problem-independent branching rule. Interestingly, the method presented in [9] makes use of a combination of DSS and CSS, just like the method presented in [10].

In this article we continue the above mentioned line of research, namely by developing the method presented in [10] further. The method is a biobjective branch-and-bound method where the integers are required to be binary, and continuous variables are allowed only in one of the two objectives. We present an approach where the criteria space is used to obtain a trivial parallelization, i.e. parallelization where no communication is needed between the processors. We test our new hybrid approach on the biobjective Travelling Salesman Problem (BITSP). Since the first proposal of the TSP problem [25], TSP has formed a testbed for Operations Research methods. The branch-and-cut algorithm was originally developed for solving the TSP problem, starting with the cutting plane approach [26] which was in the 1980s developed to solve large scale TSP problems by [27, 28], ending with the current record holder Concorde [29]. In the 1990s, the cutting plane techniques made their

way into general MIP model solvers [30], resulting in spectacular performance gains of MIP solvers [31]. Therefore, it is worth noting that the techniques applied in this paper to the BITSP problem can easily be applied to a large set of biobjective Mixed Integer Programs.

The main contributions in this paper to the field of multiobjective optimization can be summarized as follows:

1. The feasible set of nondominated points (in criterion space) are partitioned into a number of independent sets (slices). This is the basis for the development of the parallel method.
2. The slices are constructed in an "optimal" way, given a trial set of points in the feasible set in criterion space. This "optimal" partition is very important, because the solution time depends heavily on how the partition is done. We believe that the OR community can take advantage of this insight when solving other multiobjective optimization problems. One can think on the procedure as a parallel decomposition method.
3. The approach is tested on the biobjective TSP. The BITSP test problems are the full generally acknowledged BITSP test set, used in a number of other articles.
4. The method developed in this paper is generic and can be used on other biobjective (mixed integer) problems. BITSP is used to illustrate the approach. We expect that this approach can pave the way for general biobjective MIP solvers, just like the branch-and-cut algorithms originally developed for TSP were later turned into general solvers.

It should be mentioned that without partitioning the set of nondominated points in an optimal way into a number of slices, we cannot solve the biobjective TSP instances.

In §2 we give the preliminaries and in §3 we give a short overview of the branch-and-bound algorithm developed in [10]. In §4 we develop the parallel version of the branch-and-bound algorithm described in §3, and in §5 we apply it to the biobjective traveling salesman problem. In §6 we present a few results, and we conclude the paper in §7.

2. Preliminaries

In this section we give some notation and describe the biobjective mixed integer programming model for which a branch-and-bound method was developed in [10]. The branch-and-bound method will be described in Section 3.

We assume that there are only two objectives. The first objective only contains binary variables, whereas the second objective may contain both binary and continuous variables. These restrictions make it possible to represent the set of nondominated points as a discrete set in two-dimensional space.

Let $c^1, c^2 \in \mathcal{R}^n$ be two n-dimensional vectors and $h^2 \in \mathcal{R}^p$ be a p-dimensional vector. Let $z(x,y) = (z_1(x,y), z_2(x,y)) = (c^1x, c^2x + h^2y)$ be two linear objective functions with nonnegative n-dimensional integral variables $x \in \mathcal{Z}_+^n$ and nonnegative p-dimensional continuous variables $y \in \mathcal{R}_+^p$. Let A be an $m \times n$ matrix, G be an $m \times p$ matrix, and $b \in \mathcal{R}^m$.

The resulting biobjective mixed integer programming (BOMIP) model is shown in equation (1).

$$\min\{z(x,y) \mid Ax + Gy \geq b,\ x \in \mathcal{Z}_+^n,\ y \in \mathcal{R}_+^p\} \tag{1}$$

Let $\mathcal{X}$ denote the set of feasible solutions to (1), that is $\mathcal{X} = \{(x,y) \in \mathcal{Z}_+^n \times \mathcal{R}_+^p \mid Ax + Gy \geq b\}$. $\mathcal{X}$ is also referred to as the feasible set in *decision space*. Let $\mathcal{Z} = \{z \in \mathcal{R}^2 \mid z_1 = c^1 x,\ z_2 = c^2 x + h^2 y,\ (x,y) \in \mathcal{X}\}$ denote the corresponding feasible set in *criterion space*.

A point $z^2 \in \mathcal{Z}$ is *dominated* by $z^1 \in \mathcal{Z}$ if the condition in (2) is satisfied

$$z_i^1 \leq z_i^2 \quad i = 1,2, \quad \text{and} \quad z^1 \neq z^2 \tag{2}$$

If $z^2 \in \mathcal{Z}$ is *dominated* by $z^1 \in \mathcal{Z}$ we write $z^1 \prec z^2$.

The set of *efficient solutions* $\mathcal{X}_E$ in decision space is defined as

$$\mathcal{X}_E = \{(x,y) \in \mathcal{X} \mid \nexists (\bar{x},\bar{y}) \in \mathcal{X} : z(\bar{x},\bar{y}) \prec z(x,y)\},$$

and the set of *nondominated points* $\mathcal{Z}_N$ in criterion space is given by

$$\mathcal{Z}_N = \{z \in \mathcal{R}^2 \mid z = z(x,y),\ (x,y) \in \mathcal{X}_E\}.$$

We denote by z^{UL} the *upper/left* point and by z^{LR} the *lower/right* point. It is well-known [32] that these two points are nondominated. The point z^{UL} can be found by solving the lexicographic optimization problem $\text{lexmin}\{(z_1(x,y), z_2(x,y)) \mid (x,y) \in \mathcal{X}\}$, and the point z^{LR} can be found by solving the lexicographic optimization problem $\text{lexmin}\{(z_2(x,y), z_1(x,y)) \mid (x,y) \in \mathcal{X}\}$. Given z^{UL} and z^{LR} we know that if $z = (z_1, z_2)$ is a nondominated point then $z_1 \in [z_1^{UL}, z_1^{LR}]$ and $z_2 \in [z_2^{LR}, z_2^{UL}]$. In Figure 1, the grey rectangle, spanned by the two points z^{UL} and z^{LR}, is the only area where nondominated points may exist.

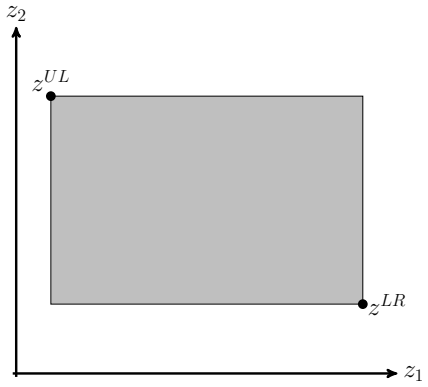


Figure 1: The criterion space

Two points in the criterion space is of particular interest. This is the ideal point $z^{ideal} = (\min_{(x,y)\in\mathcal{X}} z_1(x,y), \min_{(x,y)\in\mathcal{X}} z_2(x,y))$ and the nadir point $z^{nadir} = (\max_{(z_1,z_2)\in\mathcal{Z}_N} z_1, \max_{(z_1,z_2)\in\mathcal{Z}_N} z_2)$.

The local nadir point between two consecutive nondominated points $z^i = (z_1^i, z_2^i)$ and $z^{i+1} = (z_1^{i+1}, z_2^{i+1})$ on the Pareto front where $z_1^i < z_1^{i+1}$ and $z_2^i > z_2^{i+1}$ is the point (z_1^{i+1}, z_2^i).

In Section 3, a biobjective branch-and-bound algorithm is presented, which can find all nondominated points to problem 1. The algorithm uses the (relaxation of the) following problem:

$$\min\{z_1 + \gamma z_2 \mid Ax + Gy \geq b,\ x \in \mathcal{Z}_+^n,\ y \in \mathcal{R}_+^p\}, \tag{3}$$

where $\gamma \in]0, \infty[$ is a positive number.

2.1. Division of criteria space

In classical singleobjective branch-and-bound algorithms, the algorithm only "operates" on the decision variables. In CSS algorithms, the most typical approach is to add extra constraints in criteria space in combination with a singleobjective solve. Exactly how this is done, depends on the specific CSS algorithm in question.

The full criteria space is illustrated in Figure 1, spanned by the two lexiographic points z^{UL} and z^{LR}. Since we are only looking for efficient solutions $\mathcal{X}_E$, these can only reside (visually) in the grey area of Figure 1. In the next Section 3 we describe our biobjective branch-and-bound algorithm. We can use this algorithm to find all the efficient solutions (i.e. the Pareto Front), for the whole criteria space. But this may require an excessive use of time and memory (on the computer).

The basic hypothesis in this article is: The solution time of our branch-and-bound algorithm described in Section 3 is dependent to a certain degree on the size of the objective space to be searched (among other things).

To illustrate the hypothesis, assume that we are somehow able to find some efficient points in the criteria space, say in this case, two points, and assume we already know the lexicographic points z^{UL} and z^{LR}. This case is illustrated in Figure 2.

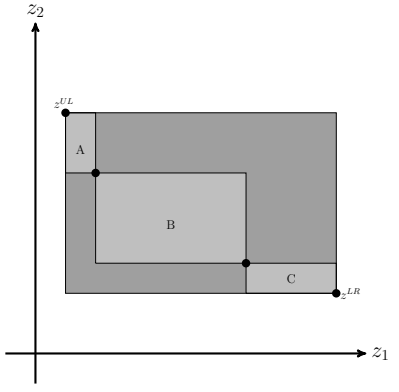


Figure 2: Division of the criterion space, using two intermediate nondominated points

In Figure 2 the two intermediate points divides the whole criteria space into 3 separate light-grey rectangles A, B and C. It is easily seen that we can only find nondominated points

in the light-grey areas. Our hypothesis then states that it is faster to execute our branch-and-bound algorithm from Section 3 three times on each of the smaller light-grey rectangles A,B and C, than to execute the branch-and-bound algorithm on the entire rectangle spanned by z^{UL} and z^{LR}.

The next question is then how to constrain the light-grey areas, which is not as simple as it seems! Assuming that all MIP solvers ultimately are singleobjective, the question is whether the objectives z_1 and z_2 are both in the objective or the coefficient in front of one of them is zero.

2.1.1. One zero coefficient

To the best of our knowledge, this case is only valid for CSS algorithms. Most of them will start with the lexicographic points z^{UL} and z^{LR} and generate the points one by one. In this case, we speculate that the criteria space reduction shown in Figure 2, has no effect on the performance. All these CSS algorithms though suffer of an important drawback: To avoid weakly nondominated points, each time a new point is found, lexicographic optimization has to be performed, i.e. first according to one criteria and then according to the other. Experimenting with this on our BITSP problem gave a surprising result: After fixing one objective, the second round of optimization on average took 3 times longer. This seemed contra-intuitive, since we constrain the feasible set for the second round. The data also showed that the second round used fewer nodes in the branch-and-bound algorithm and fewer dual-simplex iterations. The reason turns out to be that converting one objective to a constraint added a dense row to the constraints, and this lead to much slower solution time due to slower row operations in the dual-simplex [33].

2.1.2. Positive weights

When using MIP solvers, with two non-zero weights we think our hypothesis is correct and we will show empirical results in Section 6. This is the case for all DSS algorithms, but also at least one CSS algorithm [5, 6]. In [5, 6] careful adjustment of the size of the weights on the two objectives is utilized to avoid lexicographic optimization, and only using one solve to obtain a nondominated point. With positive weights, there is however a problem with the box-bounds illustrated in Figure 2: The lower-bounding constraint (in case of minimization) creates problems for the branch-and-cut algorithm. Assume that an algorithm utilizes weighted objectives for a biobjective MIP model, see the MIP model 4, where also upper and lower bounds on the criteria variables z_1 and z_2 are added, forming the reduced box area in criteria space.

$$
\begin{aligned}
&\min \alpha \cdot z_1 + (1-\alpha) \cdot z_2 \\
&\text{s.t.} \\
&z_1 = \sum_{j=1}^{n} c_j^1 x_j \\
&z_2 = \sum_{j=1}^{n} c_j^2 x_j \\
&\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, 2, \cdots, m \\
&LB_{z_1} \leq z_1 \leq UB_{z_1} \\
&LB_{z_2} \leq z_2 \leq UB_{z_2}
\end{aligned} \tag{4}
$$

For all DSS algorithms and CSS algorithms with two non-zero weights, assuming minimization on both objectives, it is always a good idea to add upper bounds on both objectives. **It is however a bad idea to add lower bounds!** The reason for this is that the lower bounds in most situations (unless the problem is very easy), makes all reduced costs of the variables zero and all dual variables zero. This "blinds" the branch-and-bound algorithm such that random branching decisions have to be made, until the lower bounds become non-binding. This is the reason we prefer a different way of bounding in criteria space: As (pizza) slices, see Figure 3

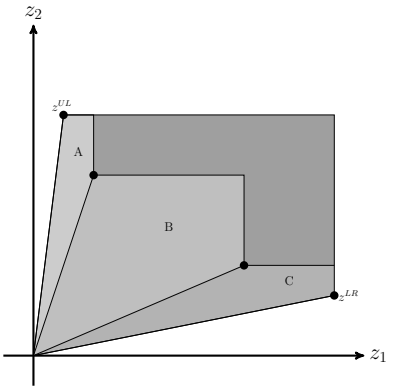


Figure 3: Division of the criterion space, using slices

In Figure 3 the three areas from the previous Figure 2 are constrained by the upper bounds, **and** two slice constraints between the two nondominated points and the origin. This is illustrated in Figure 3, where the areas A, B and C have actually been enlarged, but notice that the union of the slices contain the full A, B and C rectangles from Figure 2. This slice division also avoids "blinding" the MIP solver, and the slice constraints can actually be considered to help the MIP solver by "guiding" the algorithm in the right way.

3. Branch-and-bound algorithm

In [10] we developed a branch-and-bound algorithm for solving problem 3. The algorithm is shown in Algorithm 1.

Algorithm 1: Biobjective branch-and-bound algorithm

Input: Problem (1) and γ
Output: Set of nondominated points for problem (1)

```
1  Formulate a weighted LP (3) for problem (1)
2  Add the solution of the LP as a node to tree
3  while tree not empty do
4  |  get LP from tree
   |  /* fathom section                                  */
5  |  if LP infeasible then
6  |  |  fathom
7  |  |  continue
8  |  end
9  |  else if LP bounded (LP solution worse than the worst local nadir point) then
10 |  |  fathom
11 |  |  continue
12 |  end
   |  /* branching section                               */
13 |  if LP solution integer then
14 |  |  if not dominated: save solution in set of nondominated points
15 |  |  perform integer branching
16 |  |  add children to tree
17 |  |  continue
18 |  end
19 |  else if LP solution dominated then
20 |  |  perform Pareto branching
21 |  |  add children to tree
22 |  |  continue
23 |  end
24 |  else
25 |  |  standard variable branching
26 |  |  add children to tree
27 |  end
28 end
29 Return set of nondominated points Z_N
```

The relaxed model with the new aggregated objective function (3) can be solved with a standard LP solver. Then the main loop is entered in line 3. The algorithm does not

terminate until the tree data-structure is empty. In line 4 a new node is retrieved from the tree data-structure for consideration. First we check if the LP is infeasible (line 5), in which case it can be fathomed. Then we check if the LP solution is worse than the worst local nadir point (line 9), see Section 3.1, in which case the node can also be fathomed. If none of these two cases occur we check if the LP solution is integral (line 13), i.e. (integer) variables have integral values, in which case the node *cannot* be fathomed. Instead so-called integer-branching (line 15) is performed, see Section 3.2. If integer branching is not performed, so-called Pareto branching (line 20) is attempted, see Section 3.3. Finally, if none of the above cases are possible, standard variable branching (line 25) is performed.

3.1. LP bounded

To fathom a (fractional) LP solution it must be guaranteed that no successor of this node in the branch-and-bound tree can be part of $\mathcal{Z}_N$. The situation is illustrated in Figure 4a, where the local nadir points are marked with open squares. Two nodes z^A and z^B are shown as open circles and both are dominated. But node z^A cannot be fathomed whereas node z^B can. The reason is that the objective value of z^B is worse than the objective value of the worst local nadir point.

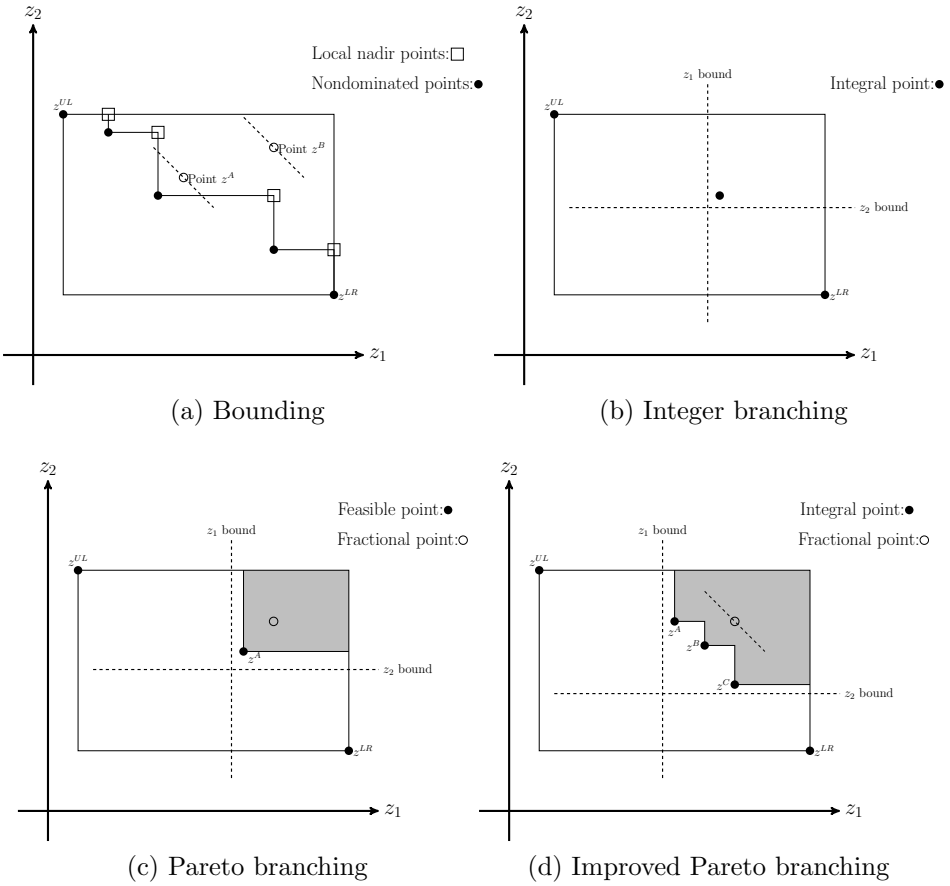
3.2. Integer branching

When a node is integral, it is still possible that later branches of the node can become part of $\mathcal{Z}_N$. Hence so-called integer branching is necessary, see Figure 4b. The integer solution is shown as the filled circle. Now two branches are created, one where a new upper bound is given for z_1 and one where a new upper bound is given for z_2. Hence the branch is set to be lower than the node z_1 value by $z_1 - \delta$. If the coefficients c_i^1 are integers the above cut is correct provided that $\delta < 1$. Also, the z_2 upper bound can be set to $z_2 - \delta$, for a small positive value of δ.

3.3. Pareto branching

The algorithm in Algorithm 1 attempts a specialized type of branching first, if the current node cannot be bounded. If the LP solution is *dominated* by a solution in the current incumbent set of nondominated points, Pareto branching can be performed, see Figure 4c. The non-integral node, shown as an open circle, is dominated by point z^A, so two new branches can be created, one where z_1 is less than the z_1 bound of point z^A and one where z_2 is less than the z_2 bound of point z^A.

The Pareto branching procedure, shown in Figure 4c, is essential for the performance of the biobjective branch-and-bound algorithm, but it can be improved, see Figure 4d. Again the LP solution is dominated by a solution, point z^B, in the current incumbent set of nondominated points, but instead of just using the z_1 and z_2 values of point z^A, we can utilize the value of the objective function in program (3), illustrated by the stapled line in the figure. Because of the objective function, we can be sure that the children of the current point can never improve the set of nondominated points between point z^A and z^B and that it can never improve the set of nondominated points between point z^B and point z^C. Hence the branch z_1 value is less than the z_1 value for point z^A and the branch z_2 is less than the z_2 value for point z^C.



(a) Bounding (b) Integer branching

(c) Pareto branching (d) Improved Pareto branching

Figure 4: **Bounding and branching**

3.4. Slicing

In our previous paper [10] the two most important algorithmic techniques which improve the performance is Pareto Branching and a technique which is called Slicing. Each particular slice is determined by two lines originating from origo. If nothing is known about the set of nondominated points the angle between the two lines defining a particular slice is the same, no matter what slice it is, as illustrated in Figure 5.

In [10] slicing was used to improve bounding, see [10] for details. In this paper we will still utilize the improved bounding, but at the same time the slices are solved in parallel, as described in Section 4. In order to have slices, two extra constraints need to be added, see below in program (5).

$$\min\{z_1 + \gamma z_2 \mid Ax + Gy \geq b,\ \alpha^{down} z_1 \leq z_2 \leq \alpha^{up} z_1,\ x \in \mathcal{Z}^n_+,\ y \in \mathcal{R}^p_+\} \tag{5}$$

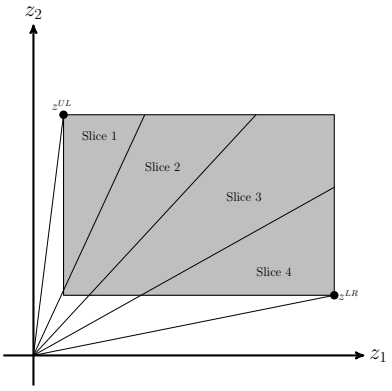


Figure 5: Visualisation of slicing

4. Criterion space parallelization

Traditional parallelization of branch-and-bound algorithms typically split the search tree into different sub-trees, each of which is solved using its own processor. The only communication which is necessary for the different processes is to exchange the value of the best incumbent solution. It should, however, be mentioned that the leading solver products, Gurobi, Cplex and Xpress use proprietary code and may hence use very different parallelization schemes than the classic approach.

Sub-tree parallelization for biobjective branch-and-bound algorithms is, however, problematic: *The set of points in $\mathcal{Z}$ that are not dominated so far is incumbent* in biobjective branch-and-bound algorithms. Given that $\mathcal{Z}_N$ may consist of a large number of points, and any new point that is not dominated by the local set of nondominated points needs to be communicated to all the other sub-tree algorithms. Hence the communication needs become significantly higher.

Below, we will present an alternative approach: Criterion space parallelization, where each slice is solved separately on different processors.

The problem of communication between the slices arises because each slice does not know the nondominated end points, i.e. the two nondominated points closest to the radial constraints. Through pre-computation we are able to find these points in two steps: First calculate the nondominated points around each radial slice constraint, see Section 4.1. Afterwards, calculate the slice nondominated end point, see Section 4.1.

4.1. Finding radial-constraint pairs of nondominated points

Given the BOMIP model (1) the set of nondominated points $\mathcal{Z}_N$ are located inside the square with extreme points $(z_1^{UL}, z_2^{LR}), (z_1^{UL}, z_2^{UL}), (z_1^{LR}, z_2^{UL})$, and (z_1^{LR}, z_2^{LR}). This square can be partitioned into two parts $\triangle_{up}$ and $\triangle_{dowm}$ by adding a radial constraint $z_2 = \alpha z_1$, where $\alpha > 0$:

$$\begin{aligned} \triangle_{up} &= \{z \in \mathcal{R}^2 \mid z_2 \geq \alpha z_1,\ z_1^{UL} \leq z_1 \leq z_1^{LR},\ z_2^{LR} \leq z_2 \leq z_2^{UL}\} \\ \triangle_{down} &= \{z \in \mathcal{R}^2 \mid z_2 \leq \alpha z_1,\ z_1^{UL} \leq z_1 \leq z_1^{LR},\ z_2^{LR} \leq z_2 \leq z_2^{UL}\} \end{aligned}$$

We may make two restrictions of problem 1. These are

$$\min\{(z(x,y) \mid c^2x + h^2y \geq \alpha c^1x,\ (x,y) \in \mathcal{X}\} \tag{6}$$

$$\min\{(z(x,y) \mid c^2x + h^2y \leq \alpha c^1x,\ (x,y) \in \mathcal{X}\} \tag{7}$$

Let $\mathcal{X}_{up}$ denote the feasible set to problem (6) and let $\mathcal{X}_{down}$ denote the feasible set to problem (7). Furthermore, let $\mathcal{Z}_{N,up}$ denote the set of nondominated points to problem (6) and let $\mathcal{Z}_{N,down}$ denote the set of nondominated points to problem (7). Clearly, $\mathcal{Z}_{N,up} \subseteq \triangle_{up}$ and $\mathcal{Z}_{N,down} \subseteq \triangle_{down}$. Because $\mathcal{X}_{up} \subseteq \mathcal{X}$ it follows that $\mathcal{Z}_N \cap \triangle_{up} \subseteq \mathcal{Z}_{N,up}$. Similarly, $\mathcal{Z}_N \cap \triangle_{down} \subseteq \mathcal{Z}_{N,down}$. Also, $\mathcal{Z}_N \subseteq (\mathcal{Z}_{N,up} \cup \mathcal{Z}_{N,down})$. However, it may be the case that $\mathcal{Z}_N \cap \triangle_{up} \subset \mathcal{Z}_{N,up}$ or $\mathcal{Z}_N \cap \triangle_{down} \subset \mathcal{Z}_{N,down}$.

We want to find two nondominated points z^A and z^B to problem (1). Point z^A should belong to $\triangle_{up}$ and be as close to the radial constraint $z_2 = \alpha z_1$ as possible. Similarly, point z^B should belong to $\triangle_{down}$ and be as close to the radial constraint $z_2 = \alpha z_1$ as possible, see Figure 6. The two points z^A and z^B are a pair of Radial Constraint Nondominated Points (RCNP). To find these points we define two optimization problems, namely one for each of the parts $\triangle_{up}$ and $\triangle_{down}$. For the part $\triangle_{up}$ we can define the model $R(\alpha, \geq)$:

$$R(\alpha, \geq) \qquad \text{lexmin}\{(z_2(x,y), z_1(x,y)) \mid (x,y) \in \mathcal{X}_{up}\}, \tag{8}$$

and for the part $\triangle_{down}$ we can define the model $R(\alpha, \leq)$:

$$R(\alpha, \leq) \qquad \text{lexmin}\{(z_1(x,y), z_2(x,y)) \mid (x,y) \in \mathcal{X}_{down}\} \tag{9}$$

Algorithm 2 shows how the two RCNP points z^A and z^B can be determined.

4.1.1. RCNP features

We would like to prove a number of different features.

Theorem 1. *The two RCNP points ($\{z^A, z^B\}$ or $\{z^A, z^{B'}\}$ or $\{z^{A'}, z^B\}$) found in Algorithm 2 belong to $\mathcal{Z}_N$.*

Proof.
Let $z^A(z^{A'})$ and $z^B(z^{B'})$ be the two points found in Algorithm 2 by solving $R(\alpha, \geq)$ and $R(\alpha, \leq)$. Notice that $z^A(z^{A'}) \in \mathcal{Z}_{N,up}$ and $z^B(z^{B'}) \in \mathcal{Z}_{N,down}$, see Figure 7.

Case 1: $z^A = z^B$.
z^A is not dominated by any points in $\mathcal{Z}_{N,up}$, and z^B is not dominated by any points in $\mathcal{Z}_{N,down}$. We can therefore conclude that z^A is not dominated by any points in $\mathcal{Z}_{N,up} \cup \mathcal{Z}_{N,down}$. It follows that $z^A \in \mathcal{Z}_N$.

In the following we assume that $z^A \neq z^B$.

Case 2: The two RCNP points ($\{z^A, z^B\}$), i.e. $z^A \not\prec z^B$ and $z^B \not\prec z^A$.
If $z_1^B < z_1^A$ it follows that $z_2^B \leq \alpha z_1^B < \alpha z_1^A \leq z_2^A$. This contradicts that $z^A \not\prec z^B$. We can

Algorithm 2: Radial constraint nondominated point pairs algorithm

Input: $R(\alpha, \geq)$ and $R(\alpha, \leq)$
Output: RCNP

Solve $R(\alpha, \geq)$. Solution $z^A = (z_1^A, z_2^A)$.
Solve $R(\alpha, \leq)$ Solution $z^B = (z_1^B, z_2^B)$.
if $z^A \prec z^B$ **then**
    // We need to re-calculate z^B
    Add the constraint $\{z \in \mathcal{R}^2 \mid z_2 \leq z_2^A - \epsilon\}$ to problem $R(\alpha, \leq)$ and solve it. Solution $z^{B'} = (z_1^{B'}, z_2^{B'})$
    return $z^A, z^{B'}$
end
else if $z^B \prec z^A$ **then**
    // We need to re-calculate z^A
    Add the constraint $\{z \in \mathcal{R}^2 \mid z_1 \leq z_1^B - \epsilon\}$ to problem $R(\alpha, \geq)$ and solve it. Solution $z^{A'} = (z_1^{A'}, z_2^{A'})$
    return $z^{A'}, z^B$
end
else
    // neither z^A nor z^B dominated by each other
    return z^A, z^B
end

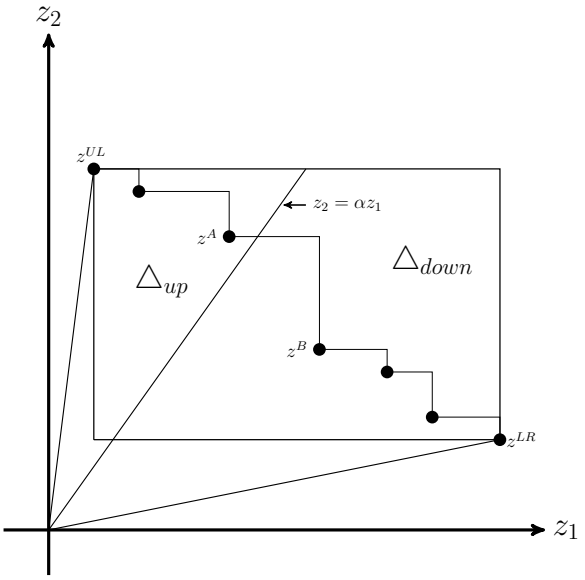


Figure 6: z^A and z^B are a pair of Radial Constraint Nondominated Points (RCNP)

conclude that $z_1^B > z_1^A$ and $z_2^B < z_2^A$. Assume that $z^B \notin \mathcal{Z}_N$. Then there exists a point $z^D \in \mathcal{Z}_N$ such that $z^D \prec z^B$. Because $\mathcal{Z}_N \subseteq (\mathcal{Z}_{N,up} \cup \mathcal{Z}_{N,down})$ it follows that $z^D \in \mathcal{Z}_{N,up}$. We know that $z_1^D \leq z_1^B$ and $z_2^D \leq z_2^B$, with at least one strict inequality. This implies $z_2^D < z_2^A$. However, z^A is the point in $\mathcal{Z}_{N,up}$ that has the smallest value of z_2. This is a contradiction, and we conclude that $z^B \in \mathcal{Z}_N$. Similarly it can be seen that $z^A \in \mathcal{Z}_N$.

Case 3: The two RCNP points ($\{z^A, z^{B'}\}$), i.e. $z^A \prec z^B$.
In this case $z_1^A \leq z_1^B$ and $z_2^A \leq z_2^B$, with at least one strict inequality. In this case B is re-calculated, see Algorithm 2. For a sufficiently small positive value of ϵ we add the constraint $\{z_2 \leq z_2^A - \epsilon\}$ to problem $R(\alpha, \leq)$. Denote the re-calculated point as $z^{B'} \in \mathcal{Z}_{N,down}$. Clearly, $z_2^{B'} < z_2^A$. This follows from the added constraint. By assumption, $z_2^A \leq z_2^B$. It follows, that $z_2^{B'} < z_2^B$. As $z^B, z^{B'} \in \mathcal{Z}_{N,down}$ it follows that $z_1^{B'} > z_1^B$. By assumption, $z_1^A \leq z_1^B$, and therefore $z_1^A < z_1^{B'}$.

Assume that $z^{B'} \notin \mathcal{Z}_N$. Then there exists a point $z^C \in \mathcal{Z}_N$ such that $z^C \prec z^{B'}$. Because $z^{B'} \in \mathcal{Z}_{N,down}$ and $\mathcal{Z}_N \subseteq (\mathcal{Z}_{N,up} \cup \mathcal{Z}_{N,down})$ it follows that $z^C \in \mathcal{Z}_{N,up}$. Also, $z_1^C \leq z_1^{B'}$ and $z_2^C \leq z_2^{B'}$, with at least one strict inequality. But then $z_2^C \leq z_2^{B'} < z_2^A$. But this contradicts that $z^A \in \mathcal{Z}_{N,up}$, because z^A is the point in $\mathcal{Z}_{N,up}$ with the smallest value of z_2. We can therefore conclude, that $z^{B'} \in \mathcal{Z}_N$.

If $z^A \notin \mathcal{Z}_N$ then there is a nondominated point $z^D \in \mathcal{Z}_N$ such that $z^D \prec z^A$, i.e. $z_1^D \leq z_1^A$ and $z_2^D \leq z_2^A$, with at least one strict inequality. Because $\mathcal{Z}_N \subseteq (\mathcal{Z}_{N,up} \cup \mathcal{Z}_{N,down})$ it follows that $z^D \in \mathcal{Z}_{N,down}$. Because $z_1^A \leq z_1^B$ and $z_2^A \leq z_2^B$, it follows that $z_1^D \leq z_1^B$ and $z_2^D \leq z_2^B$, with at least one strict inequality. This implies that $z^D \prec z^B$ and contradicts that both z^B and z^D belong to $\mathcal{Z}_{N,down}$.

Case 4: The two RCNP points ($\{z^{A'}, z^B\}$), i.e. $z^B \prec z^A$.

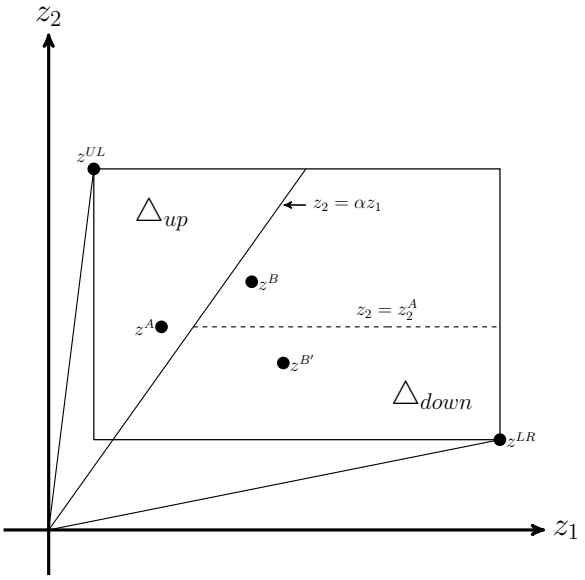


Figure 7: z^A and z^B found in Algorithm 2

Similar to case 3. □

Theorem 2. *The two RCNP points ($\{z^A, z^B\}$ or $\{z^A, z^{B'}\}$ or $\{z^{A'}, z^B\}$) found in Algorithm 2 are consecutive nondominated points in $\mathcal{Z}_N$.*

Proof.
Let $z^A(z^{A'})$ and $z^B(z^{B'})$ be the two points found in Algorithm 2 by solving $R(\alpha, \geq)$ and $R(\alpha, \leq)$. Notice that $z^A(z^{A'}) \in \mathcal{Z}_{N,up}$ and $z^B(z^{B'}) \in \mathcal{Z}_{N,down}$.

Case 1: The two RCNP points ($\{z^A, z^B\}$), i.e. $z^A \not\prec z^B$ and $z^B \not\prec z^A$.
Because $z^A \not\prec z^B$ it follows that $z_1^A < z_1^B$ (see the proof of Theorem 1, case 2). Suppose that z^A and z^B are not consecutive nondominated points in $\mathcal{Z}_N$. Then there exists a point $z^C \in \mathcal{Z}_N$ with the property

$$z_1^B > z_1^C > z_1^A \text{ and } z_2^B < z_2^C < z_2^A$$

Clearly, $z^C \in \mathcal{Z}_{N,up}$ or $z^C \in \mathcal{Z}_{N,down}$. If $z^C \in \mathcal{Z}_{N,up}$ then it contradicts that $z^A \in \mathcal{Z}_{N,up}$, because z^A is the point in $\mathcal{Z}_{N,up}$ that has the smallest value of z_2. If $z^C \in \mathcal{Z}_{N,down}$ it contradicts that $z^B \in \mathcal{Z}_{N,down}$, because z^B is the point in $\mathcal{Z}_{N,down}$ that has the smallest value of z_1.

Case 2: The two RCNP points ($\{z^A, z^{B'}\}$), i.e. $z^A \prec z^B$.
In this case point z^B is re-calculated. The new point $z^{B'} \in \mathcal{Z}_{N,down}$. As shown in Theorem 1 the two points z^A and $z^{B'}$ both belong to $\mathcal{Z}_{\mathcal{N}}$, and it was also shown that $z_1^{B'} > z_1^A$ and $z_2^{B'} < z_2^A$. If z^A and $z^{B'}$ are not consecutive nondominated points in $\mathcal{Z}_N$ then there exists a point $z^D \in \mathcal{Z}_N$ with the property

$$z_1^{B'} > z_1^D > z_1^A \text{ and } z_2^{B'} < z_2^D < z_2^A$$

Clearly, $z^D \in \mathcal{Z}_{N,up}$ or $z^D \in \mathcal{Z}_{N,down}$. If $z^D \in \mathcal{Z}_{N,up}$ then it contradicts that $z^A \in \mathcal{Z}_{N,up}$, because z^A is the point in $\mathcal{Z}_{N,up}$ that has the smallest value of z_2. If $z^D \in \mathcal{Z}_{N,down}$ it contradicts that $z^{B'} \in \mathcal{Z}_{N,down}$, because $z^{B'}$ is the point in $\mathcal{Z}_{N,down}$ that has the smallest value of z_1 under the extra condition that $z_2 < z_2^A - \epsilon$.

Case 3: The two RCNP points ($\{z^{A'}, z^B\}$), i.e. $z^B \prec z^A$.
This case can be carried out in a similar way as done in case 2. □

As mentioned in Subsection 3.4 we use slicing to improve our bounding in the biobjective branch-and-bound algorithm. In particular we want to determine the set of nondominated points within each slice in parallel (each slice is solved separately on different processors). The set of nondominated points in a particular slice $\alpha^{down} z_1 \leq z_2 \leq \alpha^{up} z_1$ is found by solving problem (5). If Algorithm 2 is used with input $R(\alpha^{up}, \geq)$ and $R(\alpha^{up}, \leq)$ we find the nondominated points z^A and z^B, and if Algorithm 2 is used with input $R(\alpha^{down}, \geq)$ and $R(\alpha^{down}, \leq)$ we find the nondominated points z^C and z^D, see Figure 8.

By construction, the nondominated points z^B and z^C are the "outer" points of the part of $\mathcal{Z}_N$ *inside* the slice defined by $\alpha^{down} z_1 \leq z_2 \leq \alpha^{up} z_1$. Hence, we can solve this slice in isolation from the other slices. Because both points are part of $\mathcal{Z}_N$ according to Theorem 1, no point exists which dominates these, and hence, no point found *outside* this slice can dominate them, hence no point outside this slice can affect the inside set of nondominated points, and the slice can be solved completely separated from the other slices.

The following lemmas are easy to verify.

Lemma 3. *If $z^A = z^C$ or $z^B = z^D$, then the interior of the slice $\alpha^{down} z_1 \leq z_2 \leq \alpha^{up} z_1$ contains no nondominated points.*

Lemma 4. *If $z^A \neq z^C$ and $z^B \neq z^D$ and if $z^B = z^C$, then the interior of the slice contains only one nondominated point, namely the point z^B.*

4.2. Heuristic construction of the slices

In Subsection 4.1 we showed how it is possible to find RCNP points which can divide the problem of finding $\mathcal{Z}_N$ into smaller parts, which can be solved independently of each other. If nothing is known about the set of nondominated points we may simply divide the rectangle spanned by z^{UL} and z^{LR}, where nondominated points may exist into, say k, slices as illustrated in Figure 5.

In our experiments with the parallel branch-and-bound algorithm we noticed that the solution time for each specific slice was very dependent on the area of the rectangle spanned by the two "outer" RCNP points within the slice (such as the points z^B and z^C illustrated in Figure 8). The smaller the area of the rectangle spanned by the two RCNP points, the faster is the set of nondominated points within the slice found. Given that we have k slices, we have k such rectangles, one in each slice. Because we are solving the problem using a parallel branch-and-bound algorithm the total solution time is very dependent upon the solution time of the slice containing the rectangle with the largest area. Therefore, we may aim at defining the k slices in such a way that the size of the area of the largest of the k

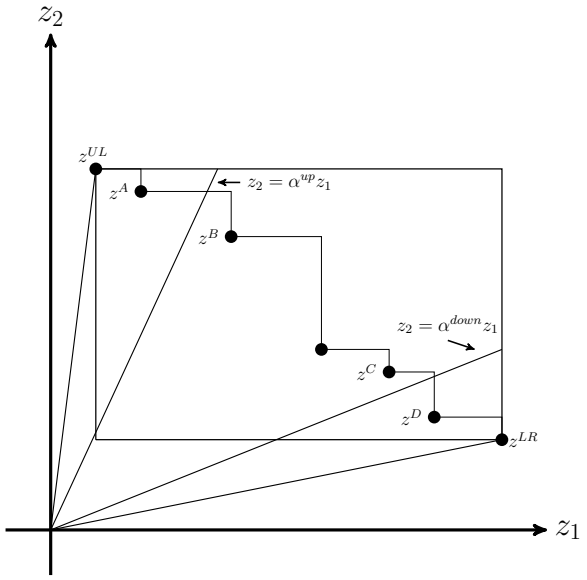


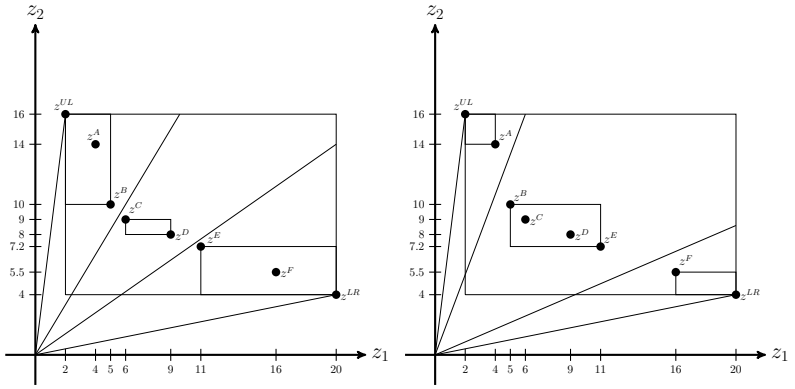
Figure 8: RCNP points for a sliced BOMIP

rectangles is as small as possible. Unfortunately, to be able to do that we need to know the full set of nondominated points.

Now suppose that we know a subset of the set of (not necessarily nondominated) points in the feasible set in criterion space that includes the two nondominated points z^{UL} and z^{LR}. Given a partition of the rectangle spanned by z^{UL} and z^{LR} into k slices, we can within each slice determine the area of the rectangle spanned by the subset of the set of nondominated solutions contained in that slice. Then we know the area of the largest of the k rectangles. The problem is then to divide the rectangle spanned by z^{UL} and z^{LR} into k slices such that the area of the largest of the k rectangles calculated in the way described above is as small as possible.

The above method is illustrated in Figure 9. Assume that we know a subset of the feasible set of points in criterion space consisting of 8 points: $z^{UL} = (2, 16), z^A = (4, 14), z^B = (5, 10), z^C = (6, 9), z^D = (9, 8), z^E = (11, 7.2), z^F = (16, 5.5), z^{LR} = (20, 4)$. We want to divide the rectangle spanned by z^{UL} and z^{LR} into 3 slices. In Figure 9a the angles between the two lines defining a slice are the same for all three slices. The rectangles within each slice determined by the known subset of nondominated points are also illustrated. The three areas, left to right, are 18, 3 and 28.8. In Figure 9b we have illustrated the optimal partitioning of the rectangles spanned by z^{UL} and z^{LR} into 3 slices. The three areas, left to right, are 4, 16.8 and 6. Notice, that the area of the largest rectangle in Figure 9b is much smaller in this particular case than the area of the largest rectangle in Figure 9a. The area of the rectangle spanned by the two points z^{UL} and z^{LR} is 216. In Figure 9a the area of the largest rectangle constitutes 13.3% of the area of the rectangle spanned by the two points z^{UL} and z^{LR}. Similarly, in Figure 9b the area of the largest rectangle constitutes 7.8% of the area of the rectangle spanned by the two points z^{UL} and z^{LR}. The sums of the three

areas in the two figures constitute 23.1% and 12.4%, respectively, of the area of the rectangle spanned by the two points z^{UL} and z^{LR}.



(a) Same angle (b) Optimal

Figure 9: **Partitioning $z^{UL} - z^{LR}$ rectangle into slices**

Assume, that we have an initial set $N = \{z^{UL}, z^1, z^2, \cdots, z^{K-2}, z^{LR}\}$ of (not necessarily nondominated) points in the feasible set in criterion space, that includes the two nondominated points z^{UL} and z^{LR}. The points are ordered increasingly according to the first coordinate, and there are K points in the initial set. The two points z^{UL} and z^{LR} may be found using lexicographic optimization, whereas the other points may have been found by using a heuristic. The problem of dividing the rectangle defined by z^{UL} and z^{LR} into k slices, such that the maximal area between the two "outer" points in a slice is as small as possible, can be solved by a shortest path algorithm. We assume that each slice contains at least two points. We define an acyclic network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodeset $\mathcal{V} = \{1, 2, \cdots, n\}$ and edgeset $\mathcal{E} = \{(i, j) \mid i, j \in \mathcal{V}\}$. The nodes are ordered in columns:

- Column 0 contains node z^{UL} and column k contains node z^{LR}.
- Column 1 contains a node corresponding to the points $\{z^1, z^2, \cdots, z^{K-2k+1}\}$.
- Column 2 contains a node corresponding to the points $\{z^3, \cdots, z^{K-2k+3}\}$.
- We continue the construction in this way. Each time we form a new column the two first nodes from the previous column is removed, and the two successor nodes of the last node in the previous column is added. Each of the columns $2, 3, \cdots, k-1$ contains $K - 2k + 1$ points.

  Column $k - 1$ contains the nodes $\{z^{2k-3}, z^{2k-1}, \cdots, z^{K-3}\}$.
- There are arcs from node z^{UL} in column 1 to all nodes in column 2. The weights of the arcs are the value of the area between the corresponding two points. For instance,

the value of the arc from z^{UL} in column 1 to node z^1 in column 2 is the value of the area of the rectangle spanned by z^{UL} and z^1.

- The value of an arc from a node z^i in column 2 to node $z^j,\ i+2 \leq j \leq K-2k+3$ in column 3 is the value of the area of the rectangle spanned by the points z^{i+1} and z^j. Similarly, the values of arcs leaving a node z^i in column a to a node $z^j,\ i+2 \leq j \leq K-2k+2a-1$ in the successive column is the area of the rectangle spanned by the points z^{i+1} and z^j.

Now consider a path $\mathcal{P}$ in the above mentioned network. It has this form $\mathcal{P} = \{z^{UL}, \bar{z}^1, \cdots, \bar{z}^{k-1}, z^{LR}\}$. So it starts at node z^{UL}, passes through one node in each of the $k-1$ columns and ends at node z^{LR}. The interpretation of the path is that $\bar{z}^1$ is the end point of slice 1, $\bar{z}^2$ is the end point of slice 2, etc.

In Figure 10 we have shown how the construction of the network is done using the 8 points $z^{UL}, z^A, \cdots, z^F, z^{LR}$ mentioned earlier in this subsection. One path is $\mathcal{P} = \{z^{UL}, z^A, z^E, z^{LR}\}$ This means that z^A is the ending point of slice 1, z^E is the ending point of slice 2, and that z^{LR} is the ending point of slice 3. Therefore, slice 1 consists of the two points $\{z^{UL}, z^A\}$, slice 2 consists of the four points $\{z^B, z^C, z^D, z^E\}$, and slice 3 consists of the two points $\{z^F, z^{LR}\}$. The values of the three arcs are $(z^{UL}, z^A) = 4, (z^A, z^E) = 16.8$ and $(z^E, z^{LR}) = 6$.

A shortest path algorithm can be applied to find the path which has the smallest maximal weight attached to it. In practice shortest path algorithms are very fast. The quality of the slices depends, however, on the quality of the initial set of points.

4.3. Optimality of the slice selection

Each slice is searched independently for nondominated points by our algorithm and our experiments in Section 6 clearly indicate that the smaller the area of the slice rectangle, the faster the algorithm terminates. The path approach described in the previous subsection finds the slice division, which leads to the minimal maximal rectangle. The division is however dependent on the quality of the points used, and in our experiments we use a set of points found using a heuristic.

5. An application to the biobjective traveling salesman problem

In this section we will show how to use the parallel branch-and-bound algorithm described in Sections 3 and 4 to solve the biobjective traveling salesman problem.

The traveling salesman problem consists in finding a shortest Hamiltonian cycle through n cities. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with nodeset $\mathcal{V} = \{1, 2, \cdots, n\}$ and edgeset $\mathcal{E} = \{(i,j) \mid i,j \in \mathcal{V}\}$. In the biobjective case (bTSP) we have two distance matrices $\{(c^1_{ij}, c^2_{ij}) \mid i,j \in \mathcal{V}\}$, and the problem is to find a cyclic permutation π of $\{1, 2, \cdots, n\}$ that minimizes $(\sum_{i=1}^{n} c^1_{i,\pi(i)}, \sum_{i=1}^{n} c^2_{i,\pi(i)})$. Here we are only interested in the symmetric case, that is $c^k_{ij} = c^k_{ji},\ i,j \in \mathcal{V},\ k = 1,2$. If we let the binary variables x_{ij} be equal to 1 if city j is visited immediately after city i the problem can be formulated as shown below.

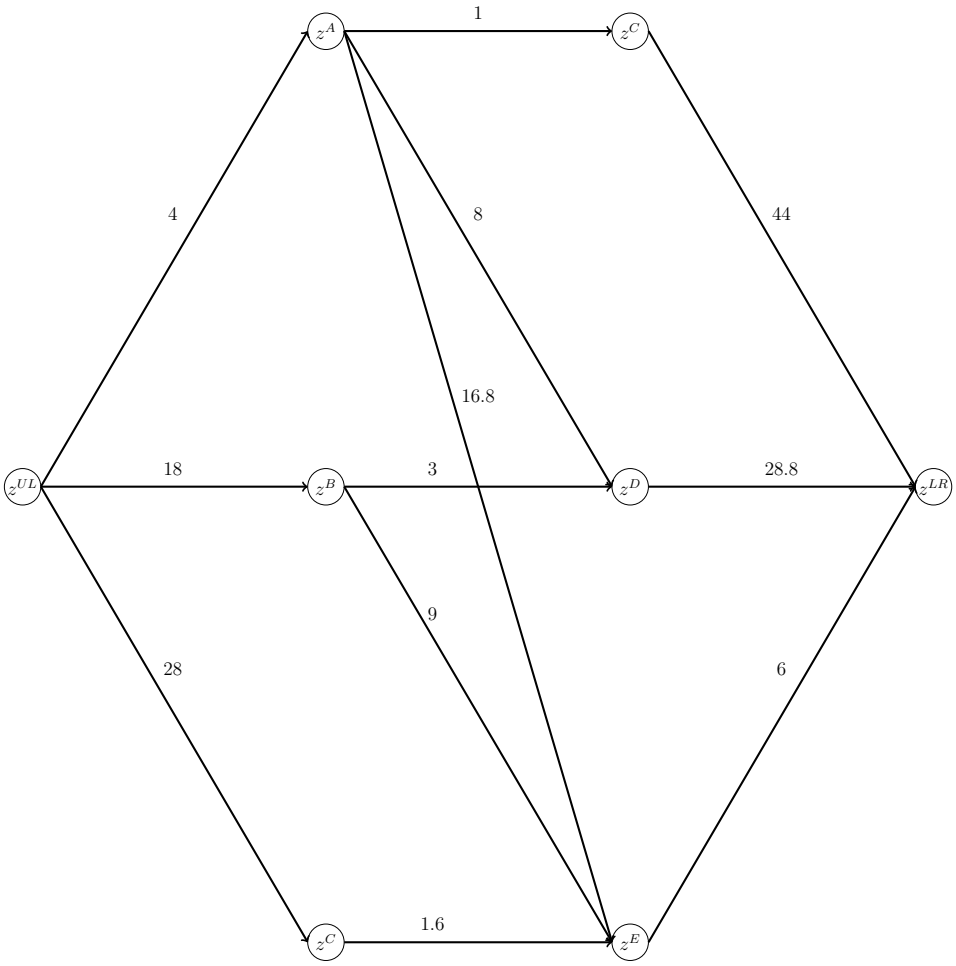


Figure 10: Network for flow algorithm.

$$
\begin{aligned}
\min \quad & z_1(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^1 x_{ij} \\
\min \quad & z_2(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^2 x_{ij} \\
\text{s.t.} \quad & \\
& \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \cdots, n \\
& \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \cdots, n \\
& \sum_{i \in S, j \in \mathcal{V} \setminus S} x_{ij} \geq 2, \quad S \subset \mathcal{V},\ 2 \leq |S| \leq |\mathcal{V}| - 2 \\
& x_{ij} \in \{0, 1\}, \quad i, j \in \mathcal{V}
\end{aligned}
\tag{10}
$$

The biobjective TSP is $\mathcal{NP}$-complete and intractable, see [32]. In particular, even though

all entries in both distance matrices are positive, the number of nondominated points may be equal to $(n-1)!$ (all Hamiltonian cycles are efficient).

5.1. Previous approaches to biobjective TSP

As the singleobjective TSP has been a testbed for a number of different singleobjective optimization techniques, multiobjective TSP could be used in a similar way to develop new multiobjective optimization techniques. However, this seems not to be the case. In the metaheuristic literature there are quite a few references to BITSP. Several approaches has been suggested, among those local search algorithms [34, 35, 36], simulated annealing [37], tabu search [38], ant colony algorithms [39] and evolutionary algorithms [40]. However, we have only been able to find **one** article [1] which solves the biobjective TSP problem exactly, i.e. find the full set of nondominated points. This article is very interesting, both because it is the first to solve the biobjective TSP and because there are similarities with our approach. In [1] the previously developed method AUGMECON2 [6] is tested on biobjective TSP problems and biobjective Set-Covering problems. To speed up solution, the AUGMECON2 algorithm is parallelized to three threads, simply by starting the same algorithm with different parameter settings. In our result Section 6 we will compare with the results in [1] and show that our approach leads to a very significant speedup.

We will briefly mention two somewhat similar biobjective problems which has been solved to optimality, namely the Multilabel Traveling Salesman Problem mTSP [8] and the Traveling Salesman Problem with profits TSPp [7].

5.2. The overall procedure

Given that program (10) contains an exponential number of sub-tour elimination constraints, typically branch-and-cut is applied. Indeed, as stated earlier, branch-and-cut was *invented* for TSP solution [28].

Our procedure for solving BITSP is divided into three steps:

- Use a heuristic procedure to find an initial set of nondominated points. The two points z^{UL} and z^{LR} are included in the initial set. To reduce the length of this paper we have used the previously found best solutions to BITSP problems instead.
- Determine the slices as described in Subsection 4.2.
- Use the parallel branch-and-bound method described in Sections 3 and 4.
  - Include cuts.

5.3. Cuts for BITSP

Notice that the set of feasible solutions to BITSP (10) is not affected by the fact that there are two objective functions. The importance of this becomes clear when one realizes that then all previously developed cuts for the traveling salesman problem can be applied in a specialized BITSP branch-and-cut algorithm.

In the past several years a number of cuts have been developed for the TSP. It is beyond the scope of this paper to describe the cut-research for TSP, and instead we refer to the book by Applegate, Bixby, Chvatal and Cook [41]. For the singleobjective TSP their Concorde [29, 42] branch-and-cut solver holds the current record for solving large-scale singleobjective TSP, by a wide margin.

Since we can apply all the previously developed cuts and the Concorde software is open-source, we simply apply the same cuts in the parallel biobjective branch-and-bound algorithm. In Algorithm 3 we describe the cutting plane procedure (procedure Cutting-Plane-Procedure()). This procedure takes as input an LP-program. Then a series of separation algorithms for finding cuts are attempted on the optimal LP solution. If one or more cuts are found it is added to the current LP problem, and the problem is reoptimized. Then we continue the search for new cuts. The procedure stops when no more cuts can be found.

We have divided the separation algorithms into two parts: Sub-tour cuts and Comb cuts. For the sub-tour cuts the separation algorithms are attempted in the following order: Sub-Tour-Connect-Cuts(), Sub-Tour-Segment-Cuts(), Sub-Tour-Shrink-Cuts() and Sub-Tour-Exact-Cuts().The Comb cuts are attempted in the order: Ghfast-Blossom(), Fast-Blossom(), Exact-Blossom(), Block-Combs(), Edge-Comb-Grower() and Cliquetree(). All the cuts and the separation algorithms to find the cuts are from the Concorde software.

As all generated cuts are valid for all the nodes in the branching tree, a separate issue arises, namely if the cuts should only be used locally (on a particular node in the branching tree) or used globally (on all the nodes in the branching tree). During our experiments we observed that when the cuts were used globally the solution time for solving the LP problems was much higher than when just using the cuts locally, even though in the latter case we might generate the same cuts several times. Furthermore, only up to about 10 % of the execution time is used for finding cuts.

5.4. Biobjective branch-and-cut algorithm for BITSP

The full biobjective branch-and-cut algorithm for BITSP is shown in Algorithm 5. It takes as input the biobjective traveling salesman problem (10) and a parameter γ. First an initial set of points are formed in line 2. Using these points k slices are constructed in line 3. Next, a weighted LP (5) for each of the k slices is formulated in line 4. The set of nondominated points within each of the k slices is found in lines 6-8, and finally the full set of nondominated points is returned in line 11. Algorithm 4 describes how to find the set of nondominated points in a particular slice. It is simply the branch-and-bound algorithm described in Algorithm 1, with the addition of the cutting plane procedure described in Algorithm 3.

6. Tests

The classic sites for TSP test problems, TSPLIB [43] and DIMACS [44] contains a large number of TSP instances of various types, all of which are singleobjective and all of which have a known optimal value. Unfortunately, such a site does not exist for BITSP problems. Instead most previous work has utilized combinations of single-objective TSP problems from

Algorithm 3: Cutting-plane-procedure()

```
   Input: An LP for BITSP
   Output: A strengthened LP
 1 Optimize LP
 2 while cuts added or first do
 3 |   if Sub-Tour-Connect-Cuts() then
 4 |   |   add cuts and break
 5 |   end
 6 |   else if Sub-Tour-Segment-Cuts() then
 7 |   |   add cuts and break
 8 |   end
 9 |   else if Sub-Tour-Shrink-Cuts() then
10 |   |   add cuts and break
11 |   end
12 |   else if Sub-Tour-Exact-Cuts() then
13 |   |   add cuts and break
14 |   end
15 |   else if Ghfast-Blossom() then
16 |   |   add cuts and break
17 |   end
18 |   else if Fast-Blossom() then
19 |   |   add cuts and break
20 |   end
21 |   else if Exact-Blossom() then
22 |   |   add cuts and break
23 |   end
24 |   else if Block-Combs() then
25 |   |   add cuts and break
26 |   end
27 |   else if Edge-Comb-Grower() then
28 |   |   add cuts and break
29 |   end
30 |   else if Cliquetree() then
31 |   |   add cuts and break
32 |   end
33 |   Re-optimize LP
34 end
```

Algorithm 4: Branch-and-Cut-for-TSP()

Input: A weighted LP (5) and initial archive
Output: Set of nondominated points for a slice

```
1  Cutting-Plane-Procedure()
2  Add the solution of the LP as a node to tree
3  while tree not empty do
4  |  get LP from tree
   |  /* fathom section                                  */
5  |  if LP infeasible then
6  |  |  fathom
7  |  |  continue
8  |  end
9  |  else if LP bounded (LP solution is worse than the worst local nadir point)
   |   then
10 |  |  fathom
11 |  |  continue
12 |  end
   |  /* branching section                               */
13 |  if LP solution integer then
14 |  |  if not dominated: save solution in set of nondominated points
15 |  |  perform integer branching
16 |  |  Cutting-Plane-Procedure()
17 |  |  add children to tree
18 |  |  continue
19 |  end
20 |  else if LP solution dominated then
21 |  |  perform Pareto branching
22 |  |  Cutting-Plane-Procedure()
23 |  |  add children to tree
24 |  |  continue
25 |  end
26 |  else
27 |  |  standard variable branching
28 |  |  Cutting-Plane-Procedure()
29 |  |  add children to tree
30 |  end
31 end
32 Return set of nondominated points for a slice
```

Algorithm 5: Biobjective branch-and-cut algorithm for BITSP

Input: Problem (10) and γ

Output: Full set of nondominated points for problem (10)

$\bar{\mathcal{Z}}_{\mathcal{N}} = \{\}$
Initial-Set-of-Points()
Generate k slices as described in Subsection 4.2
Formulate a weighted LP (5) for each of the k slices. Denote it $slice^i,\ i = 1, 2, \cdots, k$
$\bar{\mathcal{Z}}_N^i = \bar{\mathcal{Z}}_N,\ i = 1, 2, \cdots, k$
for $i = 1$ *to* k **do**
  Branch-and-Cut-for-TSP($Slice^i, \bar{\mathcal{Z}}_N^i$)
end
$\mathcal{Z}_N = \cup_{i=1}^k \bar{\mathcal{Z}}_N^i$
Remove dominated points from $\mathcal{Z}_N$
Return full set of nondominated points for problem (10)

TSPLIB [43] and DIMACS [44]. All the cost matrixes in the TSP instances have positive integer values. The big benefit of having integer costs is that rounding errors are avoided and comparisons of results become easier. Below we report on the tests of BITSP problems generated this way in three different sections.

Firstly, we compare our results with [1] in Section 6.1. There the test sets comes from two different sources:

- The 10 Lust instances [45, 46, 47]: L1 - L10.
- The Paquete instances [48, 49]: P1 - P9.

Unfortunately the L1-L10 and P1-P9 instances have overlaps, leading to 16 instances. The datasets are constructed by combining instances from [43] and from [44]. The datasets and necessary heuristic pareto fronts are downloaded from [50]. The results from these tests are given below in Section 6.1 in Table 1.

Secondly, we report the results on four 100 city BITSP datasets which are not tested in [1], generated from the Krolak/Felts/Nelson instances from TSPLIB [43], in Section 6.2.

Finally, we report the results on larger instances generated from TSPLIB [43] and DIMACS [44] in Section 6.3

The algorithms are implemented in C++ using the g++ (GCC) compiler, 4.4.7, the ILOG CPLEX 12.1.1 callable library. The biobjective branch-and-cut algorithm (Algorithm 5) is implemented using the ILOG CPLEX 12.1.1 call back routines. The cuts added, as shown in Algorithm 3, are found using separation routines from Concorde [29, 42]. The Concorde program is Open Source and can be freely downloaded. We have, however, only used the separation routines, not the entire framework.

The tests are performed on a cluster of Linux machines using up to 60 machines in parallel, each machine consisting of 2 Xeon X5550 2.67 Ghz quadcore CPU's with 8 Mb. of

Prob	10 slices		30 slices		60 slices		Opt.	Florios&Mavrotas [1]	
	Tot-Ti	Max-Time	Tot-Ti	Max-Ti	Tot-Ti	Max-Ti	P-Size	Tot-Ti	Max-Ti
kroAB100 (L1)	13893	2056	9211	436	9269	246	3332	482400	208800
kroAC100 (L2)	10633	1588	7570	420	7697	230	2458	266400	108000
kroAD100 (L3)	9240	1312	6709	342	6639	229	2351	176400	75600
kroBC100 (L4)	11929	1498	8193	439	10167	308	2752	277200	100800
kroBD100 (L5)	11961	1735	8297	504	7917	262	2657	237600	82800
kroCD100 (L6)	7833	1022	5557	274	5976	177	2044	140400	75600
euclAB100 (L7-P1)	6781	928	5036	258	5330	159	1812	147600	57600
clusAB100 (L8)	12269	2115	7051	369	7250	200	3036	187200	70200
randAB100 (L9-P4)	6555	1264	4975	303	5102	170	1707	122400	75600
mixdGG100 (L10-P7)	10261	1419	5418	307	5539	174	1848	136800	61200
euclCD100 (P2)	9330	1354	7102	399	7527	231	2268	241200	122400
euclEF100 (P3)	9129	1218	6732	426	7212	216	2530	187200	82800
randCD100 (P5)	9330	1354	5004	292	5086	185	1850	140400	57600
randEF100 (P6)	7998	1153	5466	274	5476	163	1882	158400	75600
mixedHH100(P8)	9590	1827	5897	438	5986	191	2108	126000	64800
mixedII100(P9)	6697	1080	5385	334	5316	180	1883	144000	57600
Avg.	9589	1432	6475	363	6718	207	2282	198225	86062
Speed Up Factor			1.48	3.94	1.42	6.90			

Table 1: Results comparison to Florios&Mavrotas [1].

cache and 24 Gb. 1333 Mhz DDR3 ECC ram. By using a batch system it is ensured that all tests are performed with exclusive processor rights, meaning that the programs cannot be interrupted or preempted by other processes.

All the optimal pareto fronts found are reported in the MCDMlib repository [51].

6.1. Comparison to Florios & Mavrotas [1]

In Table 1 below we compare our approach with the results from [1], on the same 16 BITSP problems. There are four different types of problems, depending on how the distance matrixes are generated: euclidean, random, mixed and clustered. The first six datasets are of the euclidean type. The first column is the name of the datasets, with the names from [1] given in brackets. In columns 2 and 3 we give total required computation time (Tot-Ti) and the maximal computation time for a slice (Max-Ti) when using 10 slices. The same results are reported in columns 4 and 5 for 30 slices and in columns 6 and 7 for 60 slices. In column 8 the size of the exact Pareto Front is given, and in columns 9 and 10, the total required computation time and the maximal required computation time is given for the approach in [1]. To run our approach, we need a heuristic solution, for these tests we use the use heuristic solutions from [50].

Comparing the results for 10, 30 and 60 slices using the averages over all 16 problems, we can see that the total time is reduced significantly going from 10 to 30 slices, but it is approximately the same going from 30 to 60 slices. The last line shows the speed up factor going from 10 slices to 30 slices and from 10 slices to 60 slices. We would like to stress that there are a number of reasons for the speed up:

1. We use significantly faster hardware: Xeon X5550 at 2.67 Ghz quadcore CPU, with CPU Mark equal to 1,252 compared, to Intel Core i3-330M at 2.13GH, with CPU Mark equal to 752 . This leads to a speed up of 67 % in our tests, compared to [1].
2. We use state-of-art TSP cuts from Concorde [29] whereas [1] utilize the branch-and-cut facility built in GAMS [52]. This probably leads to a significant speed improvement.
3. In [1] the program is parallelized into 3 threads, by running the same algorithm with different parameter settings. This in fact leads to what we term slices, but the slices

Prob	10 slices Tot-Ti	10 slices Max-Time	30 slices Tot-Ti	30 slices Max-Ti	60 slices Tot-Ti	60 slices Max-Ti	P-Size
kroAE100	9321	1368	6176	451	6024	230	2112
kroBE100	15423	2228	9079	500	9122	317	2842
kroCE100	9952	1468	6227	356	6549	235	1991
kroDE100	13349	1953	9546	532	9309	290	2751
Avg.	12011	1754	7757	459	7751	268	2424
Speed Up Factor			1.54	3.81	1.54	6.54	

Table 2: Results for four 100 city BITSP problems, not reported in [1].

Prob	#Slices	Tot-Ti	Max-Ti	P-Size
kro150ab	60	69247	1724	5136
kro200ab	60	386307	32044	8915
euclidAB300	240	2994449	35060	18388
euclidEF300	240	3024679	35343	17104

Table 3: Results for large samples

used in [1] are rather un-balanced regarding objective area, leading to un-balanced solution times. This can be seen in Figure 6 page 13 in [1].

6.2. Four 100 city problems

There are 5 different "kro" single objective datasets in the TSPLIB [43], but in [1] the kroE100 sample is not utilized. In Table 2 we give the results for the kroAE100, kroBE100, kroCE100 and kroDE100 problems. The heuristic solution needed is obtained using the heuristic from [53].

6.3. Larger BITSP problems

We can use our approach to calculate the optimal pareto fronts for larger samples and we here report the results on a number of larger test samples, see Table 3. The heuristic solution needed is obtained using the heuristic from [53].

7. Conclusion

In this article we have extended the biobjective branch-and-bound algorithm described in [10] such that it can take advantage of cuts and such that the set of nondominated points within each slice can be found in parallel. Given an initial set of points in criterion space (not necessarily nondominated points) we have presented an optimal method to construct the set of slices in a way that may speed up the solution time for the biobjective branch-and-cut procedure. We have applied the extended biobjective branch-and-cut algorithm to the biobjective traveling salesman problem. The approach presented in this article can easily be

applied to other BOMIPs where at most one of the objective functions contains continuous variables.

Acknowledgement
In memory of Professor Jens Clausen, 1951-2011, supervisor, mentor and friend.

References

[1] K. Florios, G. Mavrotas, Generation of the exact pareto set in multi-objective traveling salesman and set covering problems., Applied Mathematics and Computation 237 (2014) 1.

[2] K. Daechert, J. Gorski, K. Klamroth, An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems, Computers and Operations Research, Comput. Oper. Res, Comput Oper, Comput Oper Res, Computers and Operations Research 39 (12) (2012) 2929–2943. doi:10.1016/j.cor.2012.02.021.

[3] N. Boland, H. Charkhgard, M. Savelsbergh, A criterion space search algorithm for biobjective integer programming: The balanced box method, INFORMS Journal on Computing 27 (4) (2015) 735–754.

[4] H. W. Hamacher, C. R. Pedersen, S. Ruzika, Finding representative systems for discrete bicriterion optimization problems, Operations Research Letters 35 (3) (2007) 336–344.

[5] G. Mavrotas, Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems, Applied Mathematics and Computation 213 (2) (2009) 455 – 465. doi:http://dx.doi.org/10.1016/j.amc.2009.03.037.
URL `http://www.sciencedirect.com/science/article/pii/S0096300309002574`

[6] G. Mavrotas, K. Florios, An improved version of the augmented epsilon-constraint method (augmecon2) for finding the exact pareto set in multi-objective integer programming problems, Applied Mathematics and Computation, Appl. Math. Comput, Appl Math C, Appl Math Comput, Applied Mathematics and Computation 219 (18) (2013) 9652–9669. doi:10.1016/j.amc.2013.03.002.

[7] J.-F. Berube, M. Gendreau, J.-Y. Potvin, An exact epsilon-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits, European Journal of Operational Research 194 (1) (2009) 39–50. doi:10.1016/j.ejor.2007.12.014.

[8] N. Jozefowiez, G. Laporte, F. Semet, A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem, Informs Journal on Computing 24 (4) (2012) 554–564. doi:10.1287/ijoc.1110.0476.

[9] S. N. Parragh, F. Tricoire, Branch-and-bound for bi-objective integer programming, Optimization Online.

[10] T. R. Stidsen, K. A. Andersen, B. Dammann, A branch and bound algorithm for a class of biobjective mixed integer programs, Management Science 60 (4) (2014) 1009–1032. doi:10.1287/mnsc.2013.1802.

[11] C. Dhaenens, J. Lemesre, E. G. Talbi, K-ppm: A new exact method to solve multi-objective combinatorial optimization problems, EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 200 (1) (2010) 45–53. doi:10.1016/j.ejor.2008.12.034.

[12] J. Lemesre, C. Dhaenens, E. G. Talbi, Parallel partitioning method (ppm): A new exact method to solve bi-objective problems, COMPUTERS and OPERATIONS RESEARCH 34 (8) (2007) 2450–2462. doi:10.1016/j.cor.2005.09.014.

[13] P. Belotti, B. Soylu, M. Wiecek, A branch-and-bound algorithm for biobjective mixed-integer programs, Optimization Online, optimization-online.org.

[14] T. Vincent, F. Seipp, S. Ruzika, A. Przybylski, X. Gandibleux, Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case, Computers and Operations Research 40 (2013) 498–509.

[15] Y. Y. Haimes, L. Ladson, D. A. Wismer, Bicriterion formulation of problems of integrated system identification and system optimization (1971).

[16] E. Ulungu, J. Teghem, The two phases method: an efficient procedure to solve bi-objective combinatorial optimization problems, Foundations of Computing and Decision Sciences 20 (2) (1995) 149–165.

[17] G. Mavrotas, D. Diakoulaki, A branch and bound algorithm for mixed zero-one multiple objective linear programming, European Journal of Operational Research 107 (1998) 530–541.
[18] G. Mavrotas, D. Diakoulaki, Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming, Applied Mathematics and Computation 171 (1) (2005) 53 – 71. doi:http://dx.doi.org/10.1016/j.amc.2005.01.038.
URL http://www.sciencedirect.com/science/article/pii/S0096300305000809
[19] C. Delort, O. Spanjaard, A hybrid dynamic programming approach to the biobjective binary knapsack problem, ACM Journal of Experimental Algorithmics 18 (1) (2013) 1.1–1.21.
[20] B. Soylu, Heuristic approaches for biobjective mixed 0-1 integer linear programming problems, European Journal of Operational Research 245 (2015) 690–703.
[21] A. Raith, M. Ehrgott, A two-phase algorithm for the biobjective integer minimum cost flow problem, Computers and Operations Research 36 (6) (2009) 1945–1954.
[22] G. Mavrotas, D. Diakoulaki, Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming, Applied Mathematics and Computation 171 (2005) 53–71.
[23] M. Masin, Y. Bukchin, Diversity maximization approach for multiobjective optimization, Operations Research 56 (2) (2008) 411–424.
[24] F. Sourd, O. Spanjaard, A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem, INFORMS Journal on Computing 20 (3) (2008) 471–484.
[25] K. Menger, Bericht über ein mathematisches kolloquium 1929/30, Monatshefte für Mathematik und Physik 38 (1) (1931) 17–38. doi:10.1007/BF01700678.
[26] G. Dantzig, D. Fulkerson, S. Johnson, Solution of large scale traveling salesman problem, Operations Research 2 (4) (1954) 393–410.
[27] M. Grötschel, O. Holland, Solution of large-scale symmetric travelling salesman problems, Mathematical Programming, Series B 51 (2) (1991) 141–202.
[28] M. Padberg, G. Rinaldi, Branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1) (1991) 60–100.
[29] D. Applegate, R. Bixby, V. Chvatál, W. Cook, Concorde, http://www.tsp.gatech.edu/concorde/index.html (2003).
[30] S. Ceria, C. Cordier, H. Marchand, L. A. Wolsey, Cutting planes for integer programs with general integer variables, Mathematical programming 81 (2) (1998) 201–214.
[31] R. Bixby, E. Rothberg, Progress in computational mixed integer programminga look back from the other side of the tipping point, Annals of Operations Research 149 (1) (2007) 37–41.
[32] M. Ehrgott, Multicriteria optimization, Springer, 2005.
[33] Z. Gu, personal communication, INFORMS Annual Meeting, Nashville (2016).
[34] L. Paquete, M. Chiarandini, T. Stützle, Pareto local optima sets in the bi-objective traveling salesman problem: An experimental study, Lecture Notes in Economics and Mathematical Systems 535 (2004) 177–199.
[35] L. Paquete, T. Stützle, On the performance of local search for the biobjective traveling salesman problem., Studies in Computational Intelligence.
[36] A. Jaszkiewicz, T. Lust, Proper balance between search towards and along pareto front: biobjective tsp case study, Annals of Operations Research 254 (2017) 111–130.
[37] H. Kubotani, K. Yoshimura, Performance evaluation of acceptance probability functions for multi-objective sa, Computers & Operations Research 30 (3) (2003) 427–442.
[38] M. P. Hansen, Use of substitute scalarizing functions to guide a local search based heuristic: The case of motsp, Journal of Heuristics 6 (3) (2000) 419–431.
[39] C. Garcia-Martinez, O. Cordon, F. Herrera, A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp, European Journal of Operational Research 180 (1) (2007) 116–148. doi:10.1016/j.ejor.2006.03.041.
[40] A. Jaszkiewicz, Genetic local search for multi-objective combinatorial optimization, European Journal of Operational Research 137 (1) (2002) 50–71.

[41] D. Applegate, R. Bixby, V. Chvatál, W. Cook, The Traveling Salesman Problem: A Computational Study, Princeton Series in Applied Mathematics, Princeton University Press, 2006.
[42] D. Applegate, R. Bixby, V. Chvatál, W. Cook, The Traveling Salesman Problem, Princeton University Press, 2006.
[43] G. Reinelt, Tsplib - a travelling salesman problem library, ORSA Journal on Computing 3 (4) (1991) 376–384.
[44] D. Johnson, L. McGeoch, F. Glover, C. Rego, 8th dimacs implementation challenge.
URL http://dimacs.rutgers.edu/Challenges/TSP/
[45] T. Lust, New metaheuristics for solving moco problems: application to the knapsack problem, the traveling salesman problem and imrt optimization, Ph.D. thesis, Faculté Polytechnique de Mons (2010).
[46] T. Lust, J. Teghem, The multiobjective traveling salesman problem: A survey and a new approach., Advances in Multi-Objective Nature Inspired Computing 272 (2010) 119–141.
[47] T. Lust, J. Teghem, Two-phase pareto local search for the biobjective traveling salesman problem, Journal of Heuristics 16 (3) (2010) 475–510. doi:10.1007/s10732-009-9103-9.
[48] L. F. Paquete, Stochastic local search algorithms for multiobjective combinatorial optimization: methods and analysis, Vol. 295, IOS Press, 2005.
[49] L. Paquete, T. Stützle, Design and analysis of stochastic local search for the multiobjective traveling salesman problem, Computers and Operations Research 36 (9) (2009) 2619–2631. doi:10.1016/j.cor.2008.11.013.
[50] L. Paquete, Biobjective tsp.
URL https://eden.dei.uc.pt/ paquete/tsp/
[51] L. Nielsen, X. Gandibleux, Multi-objective optimization repository (2017).
URL https://github.com/MCDMSociety/MOrepo
[52] G. Corp., Traveling salesman problem with bch, Tech. rep., GAMS Corp. (2003).
[53] C. Ryther, Optimization of the multi-objective traveling salesman problem, Master's thesis, Technical University of Denmark (2012).